Subject: DLL and U++ type
Posted by mauro.bottizzo on Mon, 03 Apr 2006 12:45:32 GMT
View Forum Message <> Reply to Message

hi,

I am writing an DLL, using:

extern "C" { ... }

When i defining the "C" functions it's possible use U++ type, like "String", "Value", etc ...?? If no, only the poiners, like: String&, Value&, etc??

Thanks.

Subject: Re: DLL and U++ type
Posted by mirek on Mon, 03 Apr 2006 16:58:56 GMT
View Forum Message <> Reply to Message

mauro.bottizzo wrote on Mon, 03 April 2006 08:45hi,

I am writing an DLL, using:

extern "C" { ... }

When i defining the "C" functions it's possible use U++ type, like "String", "Value", etc ...?? If no, only the poiners, like: String&, Value&, etc??

Thanks.

I am not 100% sure, but I think yes, it is possible.... (extern "C" in practice just removes name mangling from your functions, means overloading will be impossible, but nothing more).

Mirek

Subject: Re: DLL and U++ type
Posted by mauro.bottizzo on Mon, 03 Apr 2006 20:58:55 GMT
View Forum Message <> Reply to Message

Well,
it's running.

Last little question about that.

I need to keep the pointer of an object, returned from my DLL function, and use it in others DLL function.

First solution that i know is to include with the DLL the header with my class declaration; but I don't like that.
Second solution that i found is to return the pointer like a integer value, named "handle", and use it like a pointer when need again. I am thinking something like when using "new", but the pointer it's get from my integer var. Example:

```
bool MyDllFunction(unsigned int handle, ...) {
  MyType *foo;
  foo = (MyType *)handle;

  MyType->....
  ...


}
```

All this now it's running well, but the question is: It's correct to use an integer for store an pointer? Or need to use an different var type??

thanks.

Subject: Re: DLL and U++ type
Posted by mirek on Mon, 03 Apr 2006 21:43:16 GMT
View Forum Message <> Reply to Message

Well, it is not 100% correct, but more or less OK. Using 'int' obviously will fail on 64-bit platform.

If you want to be more correct, use intptr_t, which is integral type guaranteed to be able to hold the pointer.

Mirek

Subject: Re: DLL and U++ type
Posted by mr_ped on Sat, 08 Apr 2006 10:41:46 GMT
View Forum Message <> Reply to Message

Another way to pass around unknown pointers is to use (void *) type. I like this because I understand it even without knowing too much about C/C++ . (intptr_t is sort of higher magic for me)

Subject: Re: DLL and U++ type
Posted by mirek on Sat, 08 Apr 2006 10:55:17 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Sat, 08 April 2006 06:41Another way to pass around unknown pointers is to use (void *) type. I like this because I understand it even without knowing too much about C/C++ . (intptr_t is sort of higher magic for me)

There is nothing magic about intptr_t, just one of standard headers (I think stdlib.h, but not quite sure) has something like

```
#ifdef *****
typedef int intptr_t;
#endif

#ifdef *****
typedef _int64 intptr_t;
#endif
```

(where ***** depends on platform, compiler etc...)

Mirek

Subject: Re: DLL and U++ type
Posted by mr_ped on Wed, 12 Apr 2006 12:01:21 GMT
View Forum Message <> Reply to Message

stdlib.h *is* higher magic for me.

I never really learned C or C++, I dislike to learn too many implementation-dependent details. When I work with some new programming language, I simply read language reference guide to learn syntax and basic commands, take some framework which inits the application, and start from there...

That allowed me to stay "virgin" clean from Win32 API, STL and many more commonly used libraries. Only one which I really did study thoroughly was DirectX, otherwise I tend to pick up as small subset as is needed to finish my work, or just do the main algorithm and let someone else to finish those API things.

Than again, I find clean assembly source more readable than some high-level templatized C++ source, so I'm sort of programmer dinosaur.

(void *) is "language reference" thing and works for me.
stdlib.h is some foreign source I never bothered to study and learn.

WARNING: I'm not suggesting anyone should follow my path, you can end up coding things which

are ready to use in some nice library. You should have at least idea what those foreign libraries are capable of - to make sure you don't reinvent the wheel all the time.
 But maybe a lesson can be learned here, it's not always that important to know every API from head... OTOH language reference is IMHO always important. Platforms and libraries do change often, language stays the same (ok, evolves a bit too, but surely not as fast as libraries). If somebody is learning Win32 API and doesn't understand C++ (language part) very good, he has very likely wrong priorities, and will suffer a lot later during development.

Subject: Re: DLL and U++ type
Posted by mirek on Thu, 13 Apr 2006 16:22:58 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Wed, 12 April 2006 08:01
(void *) is "language reference" thing and works for me.

There is a subtle but important difference between (void *) and uintptr_t - uintptr_t is _integral_ type (like int or long). E.g. you can perform full integral arithmetics with it (example: think about printing the address contained in pointer

The problem uintptr_t solves is that there is no fixed fundamental type that can be safely used to store and retrieve pointers. It is e.g. "int" or "unsigned int" on most 32-bit platforms, but that is not true on 64-bit CPUs anymore.

Thus uintptr_t solves this issue.

Mirek

Subject: Re: DLL and U++ type
Posted by gprentice on Fri, 14 Apr 2006 01:36:37 GMT
View Forum Message <> Reply to Message

Just for the record - intptr_t and uintptr_t aren't part of standard C++ or C90 and in C99 they're optional and in stdint.h header - but they appear to be "standard" extensions available in many C++ compilers and in all U++ supported compilers

Graeme
